

**课 程 实 验 报 告**

**课程名称： 算法设计**

**实验名称： 分治算法求第k小元素**

**院 系 ：计算机科学与技术**

**专业班级 ： 计卓1701**

**学 号 ： U201714487**

**姓 名 ： 蔡靖涛**

**指导教师 ： 何琨**

**2019 年 11 月 1 日**

1. **实验内容**

本次实验的内容是基于分治算法思想的求乱序数组第k小元素的算法，我所实现的具体内容是由Java语言编写使用基于分治算法思想的求乱序数组第k小元素的算法。

1. **算法描述**

我的算法可以大致分为以下几步：

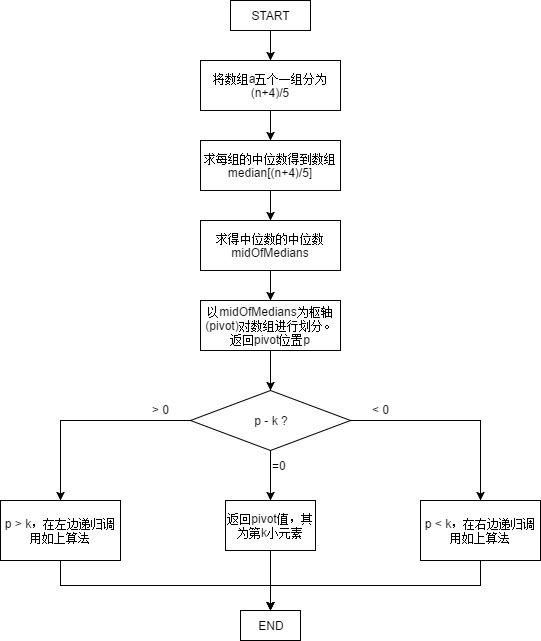
* + - 1. 将当前数组a[left,…,right]，按照排列顺序分为五个元素一组的（n+4）/5组，这里由于数组元素数目不一定是五的倍数所以最后一组可能包含少于五个元素。（n=right-left+1）
      2. 对上述每组的分别排序，取每一组的中间值形成一个新数组median[(n+4)/5]，并对这个数组排序，得到median[]的中间值midOfMedian。
      3. 对数组a依据midOfMedian进行划分，小于mOM的移到其之前，得到更新后的数组a，并且在划分时求出目前数组a里mOM的位置p
      4. 若p = k，则返回mOM的值；

若p > k，则在a[left,…,p-1]范围内递归调用如上算法；

若p < k，则在a[p+1,…,right]范围内递归调用如上算法；

如此下去直到找到第k小元素。

具体算法流程图如下：



**图2-1 算法流程图**

1. **程序设计**
   1. **数据结构定义（数据成员）**

需要操作的都是简单的int类型数组。

* 1. **核心功能实现（函数成员）**
     + 1. static int findMid(int arr[], int i, int n)

在arr[]里找到下标i开始，之后n个数内的中位数（采用这种设计是方便更改算法第一步切割数组使用的n值）。

* + - 1. static int[] swap(int[] arr, int i, int j)

交换arr[]里下标i和下标j的元素，用于在分区时移动元素。

* + - 1. static int partition(int[] arr, int l, int r, int x)

分区功能，在arr[l]到arr[r]的区域里，以x为pivot，分割数组，将比x小的放到前面，比x大的放到后面（均无序），然后返回分区后x的位置（相对l的偏移量）

* + - 1. static int kthSmallest(int arr[], int l, int r, int k)

在arr[l]到arr[r]区间内进行划分、取中、分区操作找到midOfMedians在arr[l…r]里的位置pos，随后进行判断：

若pos-l = k-1，即找到了第k小元素则返回midOfMedians;

若pos-l > k -1，返回kthSmallest(arr, l, pos-1, k);

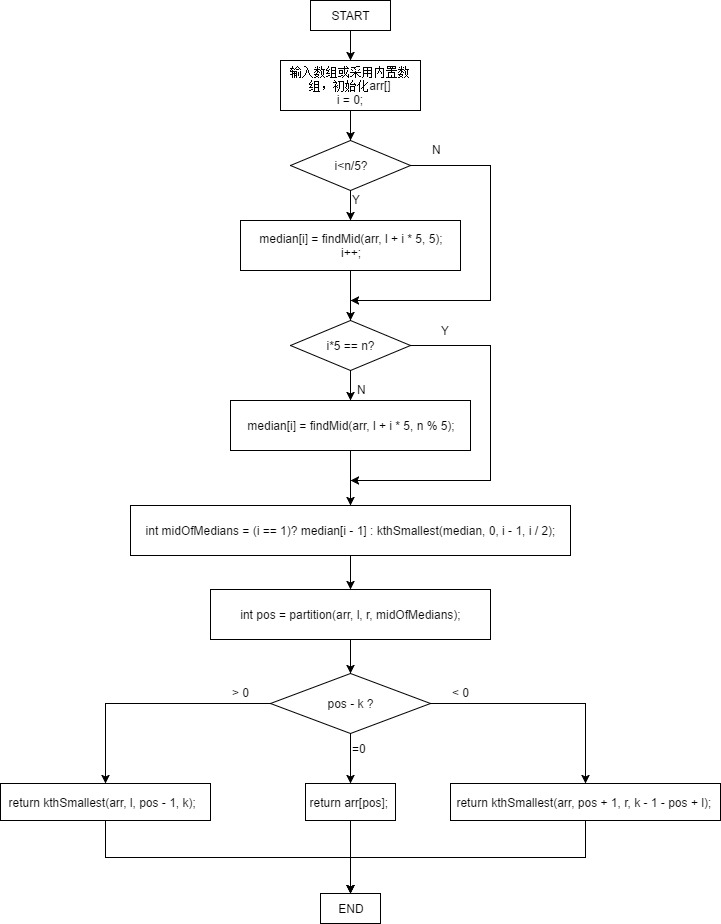
若pos-l < k -1，返回kthSmallest(arr, pos+1, r, k-1-(pos-l));

* 1. **输入输出与测试模块**

输入：采用内置数组和输入数组以及文件输入三种测试模式。输入1采用输入数组运行，输入2采用内置数组，输入3采用文件读取方式。输入数组时先输入数组元素个数随后输入元素（要求：所有元素不出现重复），然后输入k。（文件读入先读入n和k，再读入所有元素）

输出：第k小元素是…，随后询问是否再次测试，输入0退出。

具体程序流程图如下：



**图3-1 程序流程图**

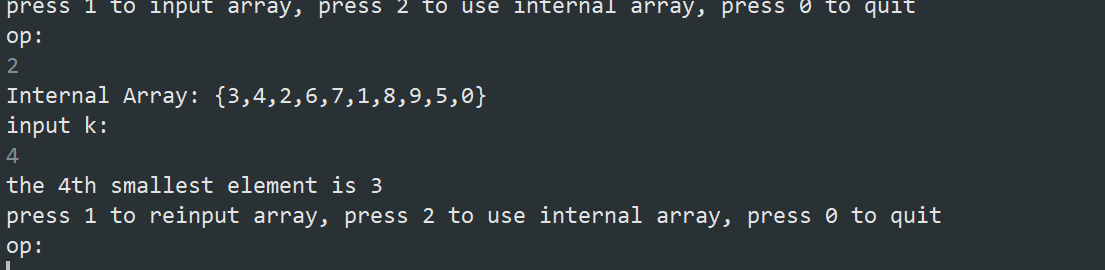
1. **结果测试**
   1. **测试环境**

该程序在运行在Windows 10系统上的IntelliJ IDEA 19.2 Community版本编译通过并成功运行（Java12）。

* 1. **测试样例与结果**
     1. **测试样例1**
        1. 输入：

采用内置数组{3, 4, 2, 6, 7, 1, 8, 9, 5, 0}，输入4（结果应为3）

* + - 1. 测试结果：

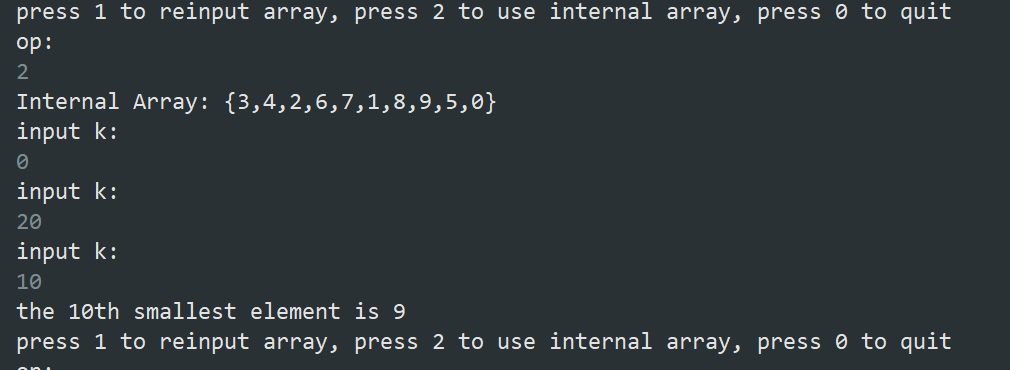


**图4-1 sample1测试结果**

* + 1. **测试样例2**
       1. 输入：

采用内置数组{3, 4, 2, 6, 7, 1, 8, 9, 5, 0}，输入0（输入非法，要求重新输入），再输入20（输入非法，要求重新输入），再输入10（结果应为9）

* + - 1. 测试结果：

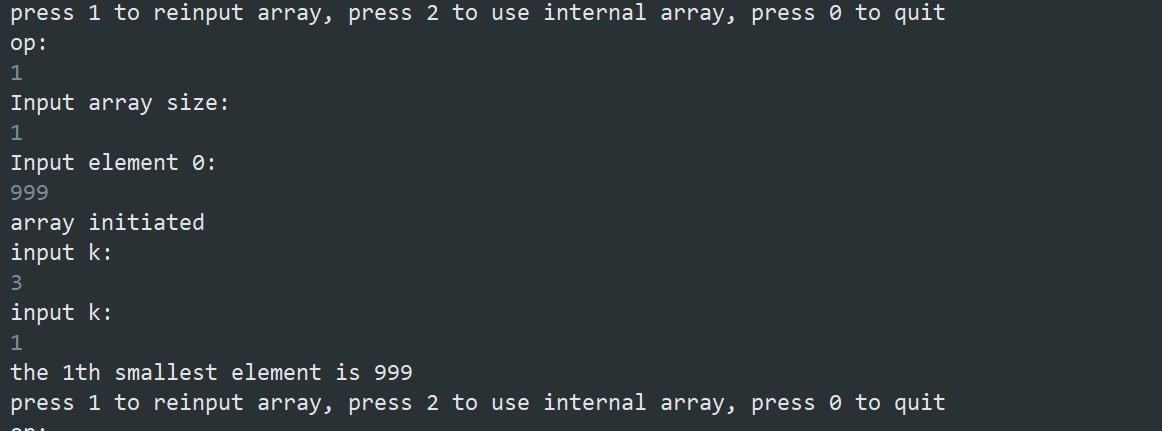


**图4-2 sample2测试结果**

* + 1. **测试样例3**
       1. 输入：

输入数组{999}，输入3（输入非法，要求重新输入），输入1（结构应为999）

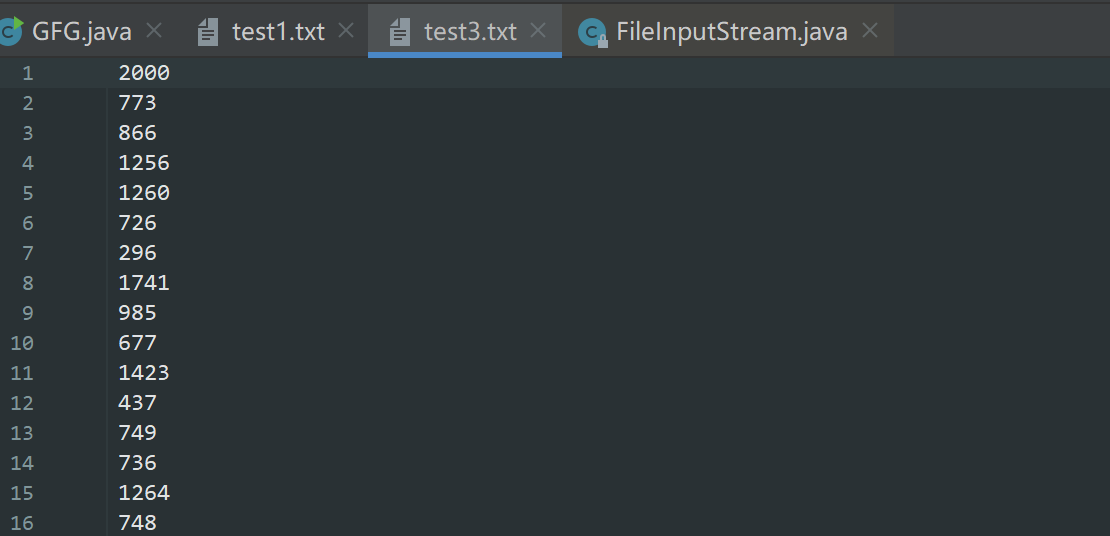
* + - 1. 测试结果：



**图4-3 sample3测试结果**

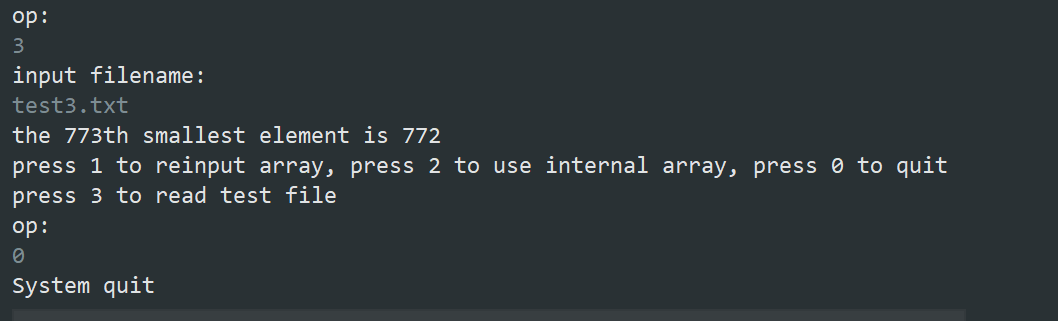
* + 1. **测试样例2**
       1. 输入：

读入文件test3.txt，这是一个由List.shuffle函数打乱了一个包含0-1999两千个Integer的列表然后输出到文件的乱序文件。包含第一行2000，第二行773（k），以及两千行每行一个整数。



**图4-4-1 sample4测试文件**

* + - 1. 测试结果：



**图4-4-2 sample4测试结果**

综合以上三次结果，可见该程序运行正常，运行结果正确。

1. **时间复杂度分析**

该算法的时间复杂度最坏为O(n)级别。

证明：该问题时间复杂度假设为T(n)

* + - 1. findMid函数在五个一组的区间内花费常数级时间查找中值，一共有(n+4)/5组，一共时间花费为O(n)。
      2. 递归调用kthSmallest求median中位数，时间复杂度T(n/5)。
      3. 分区操作一趟遍历，时间复杂度O(n)。
      4. 由于递归调用时间复杂度取决于调用次数，而我们在分割数组取中值时，找出了midOfMedians，显然其作为median的中位数，一半的medians比它大（小），而每个median所在小数组也有两个数比他们大（小），除去最后一组以及自己所在的一组，所以大于（小于）mOM的元素总数为：3\*{[(n+4)/5-2]/2} = (3/10)n - (13/6)。即最坏情况下，一趟递归一共要查找n – [(3/10)n - (13/6)] = (7/10)n + 13/6个元素。由上可得递推式：

T(n) <= O(n) + T(n/5) + O(n) + T[(7/10)n + (13/6)]

而

现在假设有一足够大的C，使得对于任意大的n，有

T(n) <= Cn/5 + 7Cn/10 + 13C/6 + O(n) <= Cn

所以总的时间复杂度为最坏O(n)级别。

1. **源代码**

package WorstCase\_On;

import java.util.\*;

import java.io.\*;

public class GFG {

//find mid elem in unsorted array

static int findMid(int arr[], int i, int n) {

Arrays.sort(arr, i, i + n); //quick sort from i to i+n

return arr[(n + i + i) / 2];

}

//swap 2 elems in an array

static int[] swap(int[] arr, int i, int j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

return arr;

}

//search int x in arr[l..r] and construct partition around x

static int partition(int[] arr, int l, int r, int x) {

int i;

//find x in array

for (i = l; i < r; i++) {

if (arr[i] == x) break;

}

//swap x to the right end

swap(arr, i, r);

//swap all elems smaller than x to the left, bigger to the right

//at last i means the number of the smaller and pos of x(start with 0)

i = l;

for (int j = l; j < r; j++) {

if (arr[j] <= x) {

swap(arr, i, j);

i++;

}

}

swap(arr, i, r);

return i; //return position of x

}

//ASSUME ALL ELEMS ARE UNIQUE

//worst case:linear time, return kth smallest in arr[l...r]

//division size : 5

static int kthSmallest(int arr[], int l, int r, int k) {

//when k legal do:

if (k > 0 && k <= r - l + 1) {

int n = r - l + 1; //number of elems in arr

int i;

//calculate median of each 5 size group and store in median[]

//all (n+4)/5 groups has 5 elems expect last group

//separately calculate mid of last group

int[] median = new int[(n + 4) / 5];

for (i = 0; i < n / 5; i++) {

median[i] = findMid(arr, l + i \* 5, 5);

}

//last group contains less than 5 elems

if (i \* 5 < n) {

median[i] = findMid(arr, l + i \* 5, n % 5);

i++;

}

int tmp = (n + 4) / 5;

//Arrays.sort(median);

//int midOfMedians = (i == 1) ? median[i - 1] : median[tmp / 2];

int midOfMedians = (i == 1) ? median[i - 1] :

kthSmallest(median, 0, i - 1, i / 2);

int pos = partition(arr, l, r, midOfMedians);

if (pos - l == k - 1)

return arr[pos]; //find the kth smallest

else if (pos - l > k - 1)

return kthSmallest(arr, l, pos - 1, k); //find in the left half, kth

else

return kthSmallest(arr, pos + 1, r, k - 1 - pos + l); //find in the right half, k-1-(pos-l) th

}

return Integer.MAX\_VALUE; //k illegal, return 0x7fffffff

}

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

int[] arrIn;

int[] arrPre;

System.out.println("Welcome to GFG");

System.out.println("press 1 to input array, press 2 to use internal array, press 0 to quit");

System.out.println("press 3 to read test file");

System.out.println("op: ");

int op;

int res, n, k;

op = in.nextInt();

while (op > 0) {

switch (op) {

case 1:

System.out.println("Input array size: ");

n = in.nextInt();

arrIn = new int[n];

for (int i1 = 0; i1 < n; i1++) {

System.out.println("Input element " + i1 + ": ");

arrIn[i1] = in.nextInt();

}

System.out.println("array initiated");

do {

System.out.println("input k: ");

k = in.nextInt();

} while (k <= 0 || k > n); //demand correct input

res = kthSmallest(arrIn, 0, n - 1, k);

System.out.println("the " + k + "th smallest element is " + res);

break;

case 2:

System.out.println("Internal Array: {3,4,2,6,7,1,8,9,5,0}");

arrPre = new int[]{3, 4, 2, 6, 7, 1, 8, 9, 5, 0};

n = arrPre.length;

do {

System.out.println("input k: ");

k = in.nextInt();

} while (k <= 0 || k > n); //demand correct input

res = kthSmallest(arrPre, 0, n - 1, k);

System.out.println("the " + k + "th smallest element is " + res);

break;

case 3:

System.out.println("input filename:");

String filename = new String();

filename = in.next();

try {

BufferedReader br = new BufferedReader(new InputStreamReader(new FileInputStream(filename)));

String data = null;

try {

data = br.readLine();

n = Integer.parseInt(data);

arrIn = new int[n];

data = br.readLine();

k = Integer.parseInt(data);

for (int j = 0; j < n; j++) {

data = br.readLine();

arrIn[j] = Integer.parseInt(data);

}

res = kthSmallest(arrIn, 0, n - 1, k);

System.out.println("the " + k + "th smallest element is " + res);

break;

} catch (IOException e) {

e.printStackTrace();

}

} catch (FileNotFoundException e) {

e.printStackTrace();

}

default:

break;

}

do {

System.out.println("press 1 to reinput array, press 2 to use internal array, press 0 to quit");

System.out.println("press 3 to read test file");

System.out.println("op: ");

op = in.nextInt();

} while (op != 0 && op != 1 && op != 2 && op != 3); //demand correct input

}

System.out.println("System quit");

}

}